



## INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification <sup>6</sup> : <b>G06F 17/50</b>	<b>A1</b>	(11) International Publication Number: <b>WO 99/12111</b> (43) International Publication Date: <b>11 March 1999 (11.03.99)</b>
(21) International Application Number: <b>PCT/US98/16436</b> (22) International Filing Date: <b>7 August 1998 (07.08.98)</b> (30) Priority Data: <b>08/919,531</b> <b>28 August 1997 (28.08.97)</b> <b>US</b> (71) Applicant: <b>XILINX, INC. [US/US]; 2100 Logic Drive, San Jose, CA 95124 (US).</b> (72) Inventor: <b>GUCCIONE, Steven, A.; 3813 McNeil Drive, Austin, TX 78727 (US).</b> (74) Agents: <b>YOUNG, Edel, M. et al.; Xilinx, Inc., 2100 Logic Drive, San Jose, CA 95124 (US).</b>		(81) Designated States: <b>JP, European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE).</b> Published <i>With international search report.          Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.</i>
(54) Title: <b>A METHOD OF DESIGNING FPGAS FOR DYNAMICALLY RECONFIGURABLE COMPUTING</b>		
<pre> graph LR     201[User Java Code] --&gt; 203((JAVA Compiler))     202[MDRC Libraries] --&gt; 203     203 --&gt; 204[Executable Code]     204 &lt;--&gt; 106[FPGA]           </pre>		
(57) Abstract <p>A method of designing FPGAs for reconfigurable computing comprises a software environment for reconfigurable coprocessor applications. This environment comprises a standard high level language compiler (i.e. Java) and a set of libraries. The FPGA is configured directly from a host processor, configuration, reconfiguration and host run-time operation being supported in a single piece of code. Design compile times on the order of seconds and built-in support for parameterized cells are significant features of the inventive method.</p>		

*FOR THE PURPOSES OF INFORMATION ONLY*

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	ES	Spain	LS	Lesotho	SI	Slovenia
AM	Armenia	FI	Finland	LT	Lithuania	SK	Slovakia
AT	Austria	FR	France	LU	Luxembourg	SN	Senegal
AU	Australia	GA	Gabon	LV	Latvia	SZ	Swaziland
AZ	Azerbaijan	GB	United Kingdom	MC	Monaco	TD	Chad
BA	Bosnia and Herzegovina	GE	Georgia	MD	Republic of Moldova	TG	Togo
BB	Barbados	GH	Ghana	MG	Madagascar	TJ	Tajikistan
BE	Belgium	GN	Guinea	MK	The former Yugoslav Republic of Macedonia	TM	Turkmenistan
BF	Burkina Faso	GR	Greece	ML	Mali	TR	Turkey
BG	Bulgaria	HU	Hungary	MN	Mongolia	TT	Trinidad and Tobago
BJ	Benin	IE	Ireland	MR	Mauritania	UA	Ukraine
BR	Brazil	IL	Israel	MW	Malawi	UG	Uganda
BY	Belarus	IS	Iceland	MX	Mexico	US	United States of America
CA	Canada	IT	Italy	NE	Niger	UZ	Uzbekistan
CF	Central African Republic	JP	Japan	NL	Netherlands	VN	Viet Nam
CG	Congo	KE	Kenya	NO	Norway	YU	Yugoslavia
CH	Switzerland	KG	Kyrgyzstan	NZ	New Zealand	ZW	Zimbabwe
CI	Côte d'Ivoire	KP	Democratic People's Republic of Korea	PL	Poland		
CM	Cameroon	KR	Republic of Korea	PT	Portugal		
CN	China	KZ	Kazakhstan	RO	Romania		
CU	Cuba	LC	Saint Lucia	RU	Russian Federation		
CZ	Czech Republic	LI	Liechtenstein	SD	Sudan		
DE	Germany	LK	Sri Lanka	SE	Sweden		
DK	Denmark	LR	Liberia	SG	Singapore		
EE	Estonia						

A METHOD OF DESIGNING FPGAS FOR  
DYNAMICALLY RECONFIGURABLE COMPUTING

5

6

BACKGROUND OF THE INVENTION

7

Field of the Invention

8

The present invention relates generally to the field of field programmable gate arrays (FPGAs) and more particularly to a method of configuring an FPGA using a host processor and a high level language compiler.

12

13

Description of the Background Art

14

15

16

17

18

In recent years, there has been an increasing interest in reconfigurable logic based processing. These systems use dynamically reconfigurable logic, such as FPGAs that can be reconfigured while in use, to implement algorithms directly in hardware, thus increasing performance.

19

20

21

22

23

24

25

By one count, at least 50 different hardware platforms (e.g., computers) have been built to investigate this novel approach to computation. Unfortunately, software has lagged behind hardware in this area. Most systems today employ traditional circuit design techniques, then interface these circuits to a host computer using standard programming languages.

26

27

28

29

30

31

32

33

34

35

36

37

38

Work done in high-level language support for reconfigurable logic based computing currently falls into two major approaches. The first approach is to use a traditional programming language in place of a hardware description language. This approach still requires software support on the host processor. The second major approach is compilation of standard programming languages to reconfigurable logic coprocessors. These approaches typically attempt to detect computationally intensive portions of code and map them to the coprocessor. These compilation tools, however, are usually tied to traditional placement and routing back-ends and have relatively slow compilation times. They also provide little or no run-time

1 support for dynamic reconfiguration.

2 In general, today's tools are based on static circuit  
3 design tools originally developed for use in circuit board  
4 and integrated circuit design. The full potential of  
5 dynamic logic is not supported by such static design tools.

6

7 SUMMARY OF THE INVENTION

8 The method of design for reconfigurable computing  
9 (MDRC) of the invention represents a novel approach to  
10 hardware/software co-design for reconfigurable logic based  
11 coprocessors. A system and method are provided for  
12 configuring an FPGA directly from a host processor. It is  
13 not necessary to store the configuration data in a file,  
14 although it can be so stored if desired. Therefore, this  
15 method is particularly suited for use with FPGAs such as  
16 reconfigurable coprocessors, which are often reconfigured  
17 "on the fly", i.e., without repowering the FPGA and  
18 sometimes while reconfiguring only a portion of the FPGA. A  
19 description of the desired functionality for the FPGA is  
20 entered using the MDRC libraries and a standard high level  
21 language such as Java™ (Java is a trademark of Sun  
22 Microsystems, Inc.). Configuration, reconfiguration and  
23 host interface software for reconfigurable coprocessors is  
24 supported in a single piece of code.

25 Since MDRC does not make use of the traditional  
26 placement and routing approach to circuit synthesis,  
27 compilation times are significantly shorter than with prior  
28 art methods, being on the order of seconds. This high-speed  
29 compilation provides a development environment which closely  
30 resembles those used for modern software development.

31 The MDRC provides a simple alternative to traditional  
32 Computer Aided Design (CAD) tool based design. In the  
33 preferred embodiment, Java libraries are used to program an  
34 FPGA device. This method has the following benefits:

35 Very fast compilation times. Because standard  
36 programming language compilers are used by this approach,  
37 compilation is as fast as the host native compiler. With

1 current Java compilers such as Microsoft's J++ 1.1 compiler  
2 compiling over 10,000 lines of code per second, compiling  
3 circuits built using MDRC will take on the order of a second  
4 to complete. This is in contrast to the hours of turnaround  
5 time in existing CAD tools.

6 Run time parameterization of circuits. Perhaps the  
7 most interesting feature of MDRC is its ability to do run-  
8 time parameterization of circuits. For instance, a constant  
9 adder, using a constant value known only at run-time, can be  
10 configured by MDRC during execution. The size of a given  
11 component may also be specified dynamically. A 5-bit adder  
12 or a 9-bit counter, for instance, can be configured at run-  
13 time. This feature has uses in areas such as adaptive  
14 filtering.

15 Object Oriented Hardware Design. Because Java is an  
16 object oriented language (i.e., a structured language in  
17 which elements are described in terms of objects and the  
18 connections between these objects), hardware designed in  
19 this language can make use of object-oriented support.  
20 Libraries constructed with MDRC may be packaged as objects  
21 and manipulated and reused like any standard software  
22 component.

23 Support for dynamic reconfiguration. The ability to  
24 dynamically configure a circuit automatically brings with it  
25 the ability to do dynamic reconfiguration. Uses for this  
26 capability are beginning to appear. For example, a portion  
27 of a dynamically reconfigurable FPGA could be configured as  
28 a multiplier that multiplies an input value by a constant,  
29 the constant being a scaling factor in a signal processing  
30 application. Using dynamic reconfiguration, this scaling  
31 factor could be changed without interrupting the function of  
32 other portions of the configured FPGA.

33 Standard software development environment. Using a  
34 standard programming language (in this case, Java) permits  
35 standard software environments to be used by circuit  
36 developers. In other words, widely available, off-the-shelf  
37 compilers such as Microsoft's J++ 1.1 compiler could be used

1 to develop circuits to be implemented in an FPGA. This  
2 capability has two immediate advantages. First, the user  
3 can continue to use whichever tool he or she is already  
4 familiar with. Secondly, and perhaps most importantly, FPGA  
5 design becomes a software development effort open to  
6 programmers. This capability could greatly expand the  
7 existing base of FPGA users.

8 Simplified host interfacing. MDRC requires a host  
9 processor to be available for executing the Java code and  
10 supplying configuration data to the FPGA. This  
11 processor/FPGA combination is a powerful coprocessing  
12 environment currently being investigated by researchers.  
13 One barrier to use of these systems is the need to interface  
14 the FPGA hardware design with the host software design.  
15 MDRC merges the software and hardware design activities into  
16 a single activity, eliminating these interfacing issues.

17 Flexibility. Because MDRC comprises a library used by  
18 a standard programming language, it may be extended, even by  
19 users. This capability provides a level of flexibility  
20 unavailable in a static design tool. Users are free to  
21 provide new libraries and library elements, or even  
22 accessories such as custom graphical user interfaces.

23 Standard device interface. One way to think of MDRC is  
24 not so much as a tool in itself, but as a standard interface  
25 to the FPGA device. This interface may be used for FPGA  
26 configuration, or it may be used to build other tools. MDRC  
27 may even be used as the basis for traditional CAD software  
28 such as placement and routing tools. Another way to think  
29 of MDRC is as the "assembly language" of the FPGA.

30 Guaranteed "safe" circuits. MDRC provides an  
31 abstraction (a software construct that provides a  
32 representation, often simplified, of the hardware) which  
33 makes it impossible to produce circuits with contention  
34 problems. This makes it impossible when using MDRC to  
35 accidentally damage or destroy the device due to a bad  
36 configuration. Such protection is highly desirable in a  
37 dynamic programming environment like MDRC, where a

1 programming error could otherwise result in permanently  
2 damaged hardware. (An incorrectly configured FPGA may  
3 inadvertently short power and ground together, destroying  
4 the device.) A side effect of this feature is that the MDRC  
5 may be used as an implementation vehicle for the emerging  
6 field of genetic algorithms.

7  
8 BRIEF DESCRIPTION OF THE DRAWINGS

9 The aforementioned objects and advantages of the  
10 present invention, as well as additional objects and  
11 advantages thereof, will be more fully understood  
12 hereinafter as a result of a detailed description of a  
13 preferred embodiment when taken in conjunction with the  
14 following drawings.

15 Figure 1 is a block diagram illustrating the prior art  
16 design flow for design of a circuit implemented in an FPGA  
17 using a reconfigurable logic coprocessor.

18 Figure 2 is a block diagram illustrating the design  
19 flow in the present invention.

20 Figure 3 is a diagram of a level 1 logic cell  
21 abstraction of the present invention.

22 Figure 3A is a diagram of an XC6200 logic cell  
23 represented by the abstraction of Figure 3.

24 Figure 4 is a diagram of a multi-bit counter according  
25 to one embodiment of the invention.

26 Figure 5 is an element definition code listing for the  
27 basic elements of the embodiment of Figure 4.

28 Figure 6A is a diagram of a toggle flip-flop cell of  
29 the embodiment of Figure 4.

30 Figure 6B is a diagram of a carry logic cell of the  
31 embodiment of Figure 4.

32 Figure 7 is a configuration code listing for the  
33 counter of Figure 4.

34 Figure 8A is a run time code for the counter of Figure  
35 4.

36 Figure 8B is an execution trace for the counter of  
37 Figure 4.

1     DETAILED DESCRIPTION OF THE DRAWINGS

2             Design of a circuit implemented in an FPGA using a  
3     reconfigurable logic coprocessor currently requires a  
4     combination of two distinct design paths, as shown in prior  
5     art Figure 1. The first and perhaps most significant  
6     portion of the effort involves circuit design using  
7     traditional CAD tools. The design path for these CAD tools  
8     typically comprises entering a design 101 using a schematic  
9     editor or hardware description language (HDL), using a  
10    netlister 102 to generate a netlist 103 for the design,  
11    importing this netlist into an FPGA placement and routing  
12    tool 104, which finally generates a bitstream file 105 of  
13    configuration data which is used to configure FPGA 106.

14            Once the configuration data has been produced, the next  
15    task is to provide software to interface the host processor  
16    to the FPGA. The user enters user code 107 describing the  
17    user interface instructions, which is then compiled using  
18    compiler 108 to produce executable code 109. The  
19    instructions in executable code 109 are then used by the  
20    processor to communicate with the configured FPGA 106. It  
21    is also known to use executable code 109 to control the  
22    configuration of FPGA 106 with bitstream file 105. This  
23    series of tasks is usually completely decoupled from the  
24    task of designing the circuit and hence can be difficult and  
25    error-prone.

26            In addition to the problems of interfacing the hardware  
27    and software in this environment, there is also the problem  
28    of design cycle time. Any change to the circuit design  
29    requires a complete pass through the hardware design tool  
30    chain (101-106 in Figure 1). This process is time  
31    consuming, with the place and route portion of the chain  
32    typically taking several hours to complete.

33            Finally, this approach provides no support for  
34    reconfiguration. The traditional hardware design tools  
35    provide support almost exclusively for static design. It is  
36    difficult to imagine constructs to support run-time



1 reconfiguration in environments based on schematic or HDL  
2 design entry.

3 In contrast, the MDRC environment comprises a library  
4 of elements which permit logic and routing to be specified  
5 and configured in a reconfigurable logic device. By making  
6 calls to these library elements, circuits may be configured  
7 and reconfigured. Additionally, host code may be written to  
8 interact with the reconfigurable hardware. This permits all  
9 design data to reside in a single system, often in a single  
10 Java source code file.

11 In addition to greatly simplifying the design flow, as  
12 shown in Figure 2, the MDRC approach also tightly couples  
13 the hardware and software design processes. Design  
14 parameters for both the reconfigurable hardware and the host  
15 software are shared. This coupling provides better support  
16 for the task of interfacing the logic circuits to the  
17 software.

18 As shown in Figure 2, entering and compiling an FPGA  
19 circuit using the MDRC method requires many fewer steps than  
20 in the prior art method of Figure 1. User code 201, in this  
21 embodiment Java code, is entered. This code includes not  
22 just instructions describing the user interface and the  
23 configuration process, but also a high-level description of  
24 the desired FPGA circuit. This circuit description  
25 comprises calls to library elements (function calls) in MDRC  
26 libraries 202. In one embodiment, these cells can be  
27 parameterized. Java compiler 203 combines the circuit  
28 descriptions from MDRC libraries 202 with the instructions  
29 from user code 201 to generate executable code 204.  
30 Executable code 204 includes not only user interface  
31 instructions, as in executable code 109 of Figure 1, but  
32 also configuration instructions. When using MDRC, the  
33 bitstream need not be stored as a file; if desired the  
34 configuration data can be directly downloaded to FPGA 106 by  
35 executable code 204. This technique is particularly useful  
36 in reconfigurable computing, i.e., when using a

1       reconfigurable FPGA as a coprocessor to perform a series of  
2       different computations for a microprocessor.

3  
4       The MDRC Abstraction

5             MDRC takes a layered approach to representing the  
6       reconfigurable logic. At the lowest (most detailed) layer,  
7       called Level 0, MDRC supports all accessible hardware  
8       resources in the reconfigurable logic. Extensive use of  
9       constants and other symbolic data makes Level 0 usable, in  
10      spite of the necessarily low level of abstraction.

11            The current platform for the MDRC environment is the  
12      XC6200DS Development System manufactured by Xilinx, Inc. the  
13      assignee of the present invention. The XC6200DS Development  
14      System comprises a PCI board containing a Xilinx XC6216  
15      FPGA. In the XC6200 family of FPGAs, Level 0 support  
16      comprises abstractions for the reconfigurable logic cells  
17      and all routing switches, including the clock routing. The  
18      code for Level 0 is essentially the bit-level information in  
19      the XC6200 Data Sheet coded into Java. (The "XC6200 Data  
20      Sheet" as referenced herein comprises pages 4-251 to 4-286  
21      of the Xilinx 1996 Data Book entitled "The Programmable  
22      Logic Data Book", published September 1996, available from  
23      Xilinx, Inc., 2100 Logic Drive, San Jose, California 95124.  
24      (Xilinx, Inc., owner of the copyright, has no objection to  
25      copying these and other pages referenced herein but  
26      otherwise reserves all copyright rights whatsoever.)

27            While Level 0 provides complete support for configuring  
28      all aspects of the device, it is very low level and may be  
29      too tedious and require too much specialized knowledge of  
30      the architecture for most users. Although this layer is  
31      always available to the programmer, it is expected that  
32      Level 0 support will function primarily as the basis for the  
33      higher layers of abstraction. In this sense, Level 0 is the  
34      "assembly language" of the MDRC system.

35            Above the Level 0 abstraction is the Level 1  
36      abstraction. This level of abstraction permits simpler

1 access to logic definition, clock and clear routing, and the  
2 host interface.

3 The most significant portion of the Level 1 abstraction  
4 is the logic cell definition. Using the logic cell  
5 definition, one logic cell in the XC6200 device can be  
6 configured as a standard logic operator. In one embodiment,  
7 AND, NAND, OR, NOR, XOR, XNOR, BUFFER and INVERTER  
8 combinational logic elements are supported. These elements  
9 may take an optional registered output. Additionally, a D  
10 flip-flop and a register logic cell are defined. In one  
11 embodiment, a latch cell is defined instead of or in  
12 addition to the flip-flop element. All of these logic  
13 operators are defined exclusively using MDRC level 0  
14 operations, and hence are easily extended.

15 Figure 3 is a diagram of the Level 1 logic cell  
16 abstraction. Outputs Nout, Eout, Sout, Wout correspond to  
17 the outputs of the same names in the XC6200 logic cell, as  
18 pictured on page 4-256 of the XC6200 data sheet. The XC6200  
19 logic cell is also shown in Figure 3A herein. Input Sin of  
20 Figure 3 corresponds to input S of the logic cell of Figure  
21 3A, input Win corresponds to input W, Nin to N, and Ein to  
22 E. The Level 1 abstraction shown in Figure 3 is a  
23 simplified representation of the XC6200 logic block. In  
24 this embodiment, for example, inputs S4, W4, N4, and E4 are  
25 not supported in the Level 1 abstraction, although they are  
26 supported in the Level 0 abstraction. The Logic block and  
27 flip-flop shown in Figure 3 signify the circuits available  
28 in one XC6200 logic cell. Inputs A, B, and SEL in Figure 3  
29 (corresponding to inputs X1, X2, and X3 of Figure 3A) are  
30 the inputs to the Logic block; they can be mapped to any of  
31 logic cell inputs Sin, Win, Nin, and Ein. The circuits  
32 available in one logic cell differ in other FPGA devices.

33 In addition to the logic cell abstraction, the clock  
34 routing is abstracted. Various global and local clock  
35 signals (such as Clk and Clr in Figure 3) may be defined and  
36 associated with a given logic cell.

1           A third portion of the MDRC Level 1 abstraction is the  
2   register interface. In the XC6200 device, columns of cells  
3   may be read or written via the bus interface, the columns of  
4   cells thus forming read/write registers. The Register  
5   interface allows registers to be constructed and accessed  
6   symbolically.

#### 7 8   An Example

9           Figure 4 shows a simple counter designed for an XC6200  
10   device, based on toggle flip-flops 402 and carry logic 401  
11   using the Level 1 abstraction. In less than 30 lines of  
12   code, the circuit is described and configured, and clocking  
13   and reading of the counter value is performed. In addition,  
14   the structure of this circuit permits it to be easily  
15   packaged as a parameterized object, with the number of bits  
16   in the counter being set via a user-defined parameter. Such  
17   an object-based approach would permit counters of any size  
18   to be specified and placed at any location in the XC6200  
19   device. Once implemented, the counter of Figure 4 could  
20   also be placed in a library of parameterized macrocells.

21           The implementation process is fairly simple. First,  
22   the logic elements required by the circuit are defined.  
23   These circuit element definitions are abstractions and are  
24   not associated with any particular hardware implementation.

25           Once these logic elements are defined, they may be  
26   written to the hardware, configuring the circuit. Once the  
27   circuit is configured, run time interfacing of the circuit,  
28   usually in the form of reading and writing registers and  
29   clocking the circuit, is performed. If the application  
30   demands it, the process may be repeated, with the hardware  
31   being reconfigured as necessary.

32           The counter example contains nine basic elements. Five  
33   basic elements provide all necessary support circuitry to  
34   read, write, clock and clear the hardware. The remaining  
35   basic elements are used to define the counter circuit  
36   itself. These elements are best seen by looking at Figure 5  
37   in conjunction with Figure 4. Figure 5 gives the MDRC code

1       for describing the basic elements. The pci6200 object  
2       passed to each of the two register definitions is the  
3       hardware interface to the XC6200DS PCI board.

4       The support circuitry includes two registers which  
5       simply interface the circuit to the host software. These  
6       two registers are used to read the value of the counter  
7       ("Register counterReg" in Figure 5) and to toggle a single  
8       flip-flop 404, producing the local clock ("Register  
9       clockReg" in Figure 5). To support the flip-flops in the  
10      XC6200 device, clock and clear (reset) inputs must also be  
11      defined. The global clock ("ClockMux globalClock" in Figure  
12      5) is the system clock for the device and must be used as  
13      the clock input to any writable register. In this circuit,  
14      the flip-flop which provides the software-controlled local  
15      clock must use the global clock. The local clock ("ClockMux  
16      localClock" in Figure 5) is the output of the software  
17      controlled clock, and must be routed to the toggle flip-  
18      flops which make up the counter. Finally, all flip-flops in  
19      the XC6200 device need a clear input ("ClearMux clear" in  
20      Figure 5). In this embodiment, the clear input to all flip-  
21      flops is simply set to logic zero (GND).

22      The first logic element in the counter circuit is the  
23      clock ("Logic clock" in Figure 5). This element is just a  
24      single bit register 404 (Figure 4) which is writable by the  
25      software. Toggling register 404 via software control  
26      produces clock Local\_clock for the counter circuit. The  
27      next counter circuit element is a toggle flip-flop such as  
28      flip-flop 402, ("Logic tff" in Figure 5). This flip-flop is  
29      defined as having an input coming from the west. (According  
30      to the standard XC6200 data sheet nomenclature, the names  
31      Logic.EAST and Ein denote an east-bound signal, i.e., a  
32      signal coming from the west.) The toggle flip-flop element  
33      provides the state storage for the counter. Next, the carry  
34      logic element 401 for the counter ("Logic carry" in Figure  
35      5) is simply an AND-gate with inputs from the previous stage  
36      carry logic and the output of the current stage toggle flip-  
37      flop. The carry element generates the "toggle" signal for

1     the next stage of the counter. Figures 6A and 6B are  
2     graphical representations of the flip-flop and carry logic  
3     cells, respectively, in an XC6200 device. Finally, a  
4     logical "one" or VCC cell ("Logic one" in Figure 5, block  
5     403 in Figure 4) is implemented for the carry input to the  
6     first stage of the counter.

7             Once this collection of abstract elements is defined,  
8     they may be instantiated anywhere in the XC6200 cell array.  
9     This instantiation is accomplished by making a call to the  
10    write() function associated with each object. This function  
11    takes a column and row parameter which define the cell in  
12    the XC6200 device to be configured. Additionally, the  
13    hardware interface object is passed as a parameter. In this  
14    case, all configuration is done to pci6200, a single  
15    XC6200DS PCI board.

16            An example of this instantiation is shown in Figure 7,  
17    which instantiates the elements for the counter of Figure 4.  
18    The code in Figure 7 performs all necessary configuration.  
19    In the for() loop, the carry cells (401 in Figure 4) are in  
20    one column with the toggle flip-flops tff (402 in Figure 4)  
21    in the next column. A local clock and a clear are attached  
22    to each toggle flip-flop tff. The relative location of  
23    these cells is shown in Figure 4.

24            Below the for() loop, a constant "1" is set as the  
25    input to the carry chain (403 in Figure 4). Next, the  
26    software-controlled clock (Local\_clock in Figure 4) is  
27    configured. This is the clock object, with its localClock  
28    routing attached to the toggle flip-flops tff of the  
29    counter. Finally, the global clock is used to clock the  
30    software-controlled local clock. In some embodiments, the  
31    clock and clear basic elements are not required; in this  
32    embodiment their presence is necessary to support the XC6200  
33    architecture.

34            Once the circuit is configured, it is a simple matter  
35    to read and write the Register objects via the get() and  
36    set() functions, respectively. In Figure 8A, the clock is  
37    toggled by alternately writing "0" and "1" to the clock

1 register (404 in Figure 4). The counter register (not  
2 shown) is used to read the value of the counter (outputs  
3 COUNT[0], COUNT[1], COUNT [2], etc.). Figure 8B shows an  
4 actual trace of the execution of this code running on the  
5 XC6200DS development system.

6

## 7 Conclusions

8 While this example is a simple one for demonstration  
9 purposes, it makes use of all the features of MDRC. These  
10 features include register reads and writes, as well as  
11 features such as software-driven local clocking. Other more  
12 complex circuits have also been developed using MDRC. More  
13 complex circuits are built using the same basic features;  
14 the primary difference is in the size of the code.

15 MDRC provides a simple, fast, integrated tool for  
16 reconfigurable logic based processing. MDRC is currently a  
17 manual tool, since it is desirable for the programmer to  
18 exercise tight control over the placement and routing of  
19 circuits for reconfigurable computing. However, MDRC  
20 provides very fast compilation times in exchange for the  
21 manual design style. The compile times necessary to produce  
22 these circuits and run-time support code is on the order of  
23 seconds, many orders of magnitude faster than the design  
24 cycle time of traditional CAD tools. This unusual speed  
25 permits development in an environment that is similar to a  
26 modern integrated software development environment.

27 Additionally, the object-oriented nature of Java permits  
28 libraries of parameterized cells to be built. This feature  
29 could significantly increase the productivity of MDRC users.

30 MDRC may be used as a basis for a traditional graphical  
31 CAD tool. This approach would be useful for producing  
32 static circuits.

33 The above text describes the MDRC in the context of  
34 FPGAs used for dynamically reconfigurable computing, such as  
35 the Xilinx XC6200 family of FPGAs. However, the invention  
36 can also be applied to other FPGAs and other software  
37 programmable ICs not used for dynamically reconfigurable

1       computing.

2           Those having skill in the relevant arts of the  
3       invention will now perceive various modifications and  
4       additions which may be made as a result of the disclosure  
5       herein. Accordingly, all such modifications and additions  
6       are deemed to be within the scope of the invention, which is  
7       to be limited only by the appended claims and their  
8       equivalents.

9



1        CLAIMS

2        What is claimed is:

3

4        1. A method of configuring a field programmable gate array  
5        (FPGA), the FPGA being connected to a host processor for  
6        configuration thereby; the method comprising the steps of:

7            a) programming the host processor with instructions in  
8        a high level programming language;

9            b) instantiating elements from a library of elements  
10       compatible with the high level programming language;

11           c) providing a compiler to the host processor for  
12       generating executable code in response to the programmed  
13       instructions and the instantiated library elements; and

14           d) configuring the FPGA from the host processor in  
15       response to the executable code.

16

17       2. The method recited in Claim 1 wherein the FPGA is used  
18       for dynamically reconfigurable computing.

19

20       3. The method recited in Claim 1 or Claim 2 wherein the  
21       high level language is Java.

22

23       4. The method recited in Claim 1 or Claim 2 wherein the  
24       library comprises combinational logic elements.

25

26       5. The method recited in Claim 1 or Claim 2 wherein the  
27       library comprises flip-flop elements.

28

29       6. The method recited in Claim 1 or Claim 2 wherein the  
30       library comprises latch elements.

31

32       7. The method recited in Claim 1 or Claim 2 further  
33       comprising the step of:

34            e) using the library elements to generate a  
35       parameterized cell.

36

37

- 1       8. The method recited in Claim 7 wherein the cell is a  
2       counter parameterized by the number of bits in the counter.  
3
- 4       9. A method of configuring a field programmable gate array  
5       (FPGA) for dynamically reconfigurable computing; the method  
6       comprising the steps of:  
7       a) programming the host processor with instructions in  
8       a high level language;  
9       b) providing a compiler running on the host processor  
10      for generating executable code in response to the  
11      instructions; and  
12      c) connecting the host processor to the FPGA for  
13      dynamic reconfiguration programming of the FPGA by the host  
14      processor via the executable code.  
15
- 16      10. The method recited in Claim 9 wherein the high level  
17      language is Java.  
18
- 19      11. The method recited in Claim 9 further comprising the  
20      step of:  
21      d) instantiating elements from a library of elements  
22      compatible with the compiler.  
23
- 24      12. The method recited in Claim 11 wherein the library  
25      comprises combinational logic elements.  
26
- 27      13. The method recited in Claim 11 wherein the library  
28      comprises flip-flop elements.  
29
- 30      14. The method recited in Claim 11 wherein the library  
31      comprises latch elements.  
32  
33  
34  
35  
36  
37

1/7

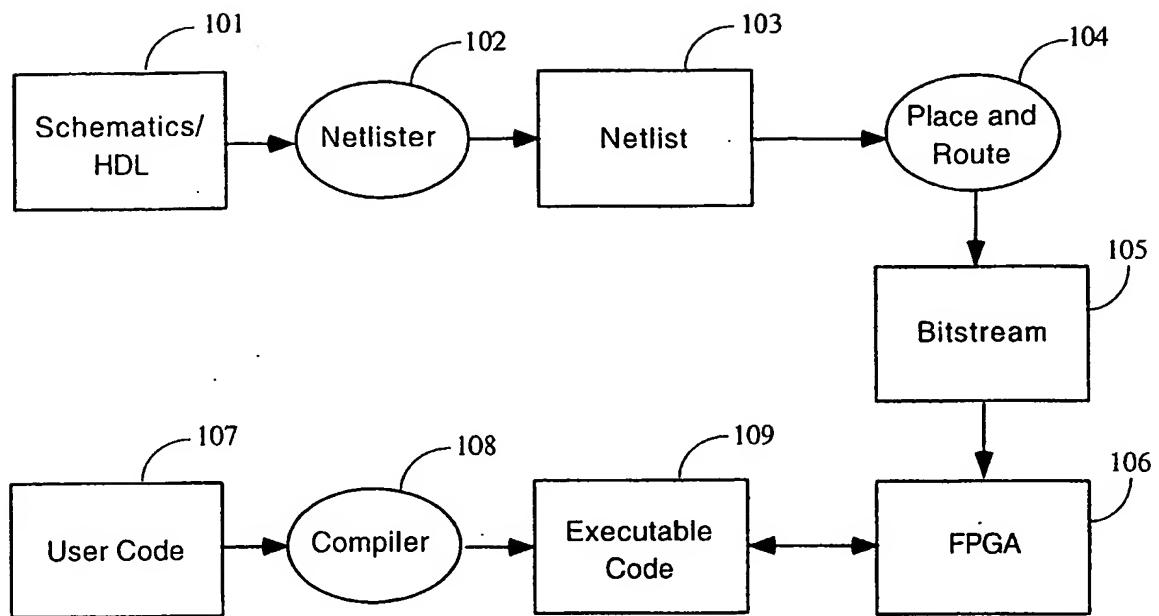


FIG. 1 (Prior Art)

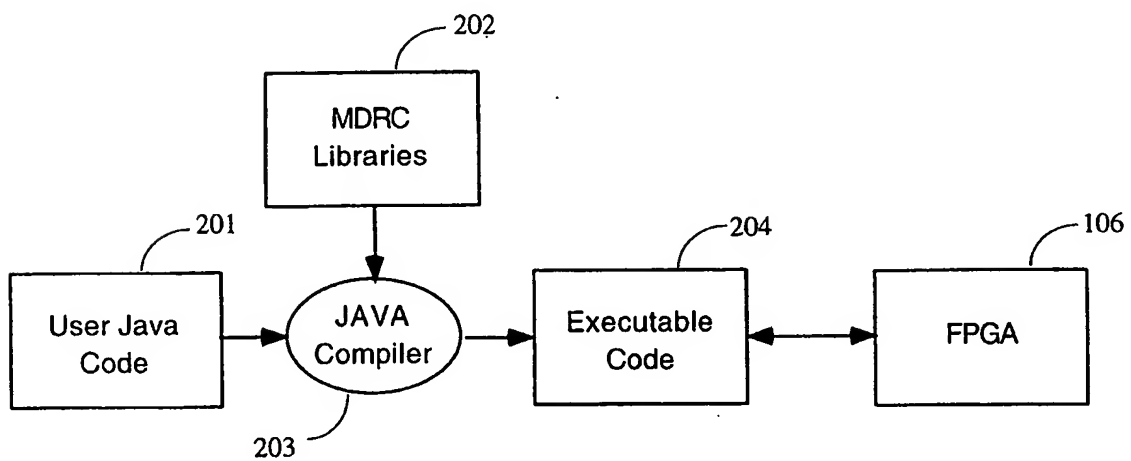


FIG. 2

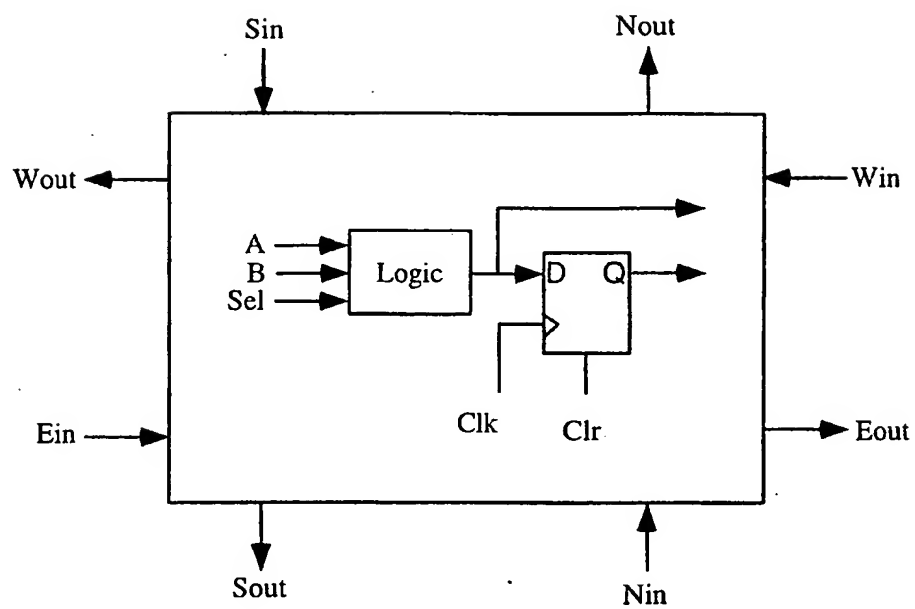


FIG. 3

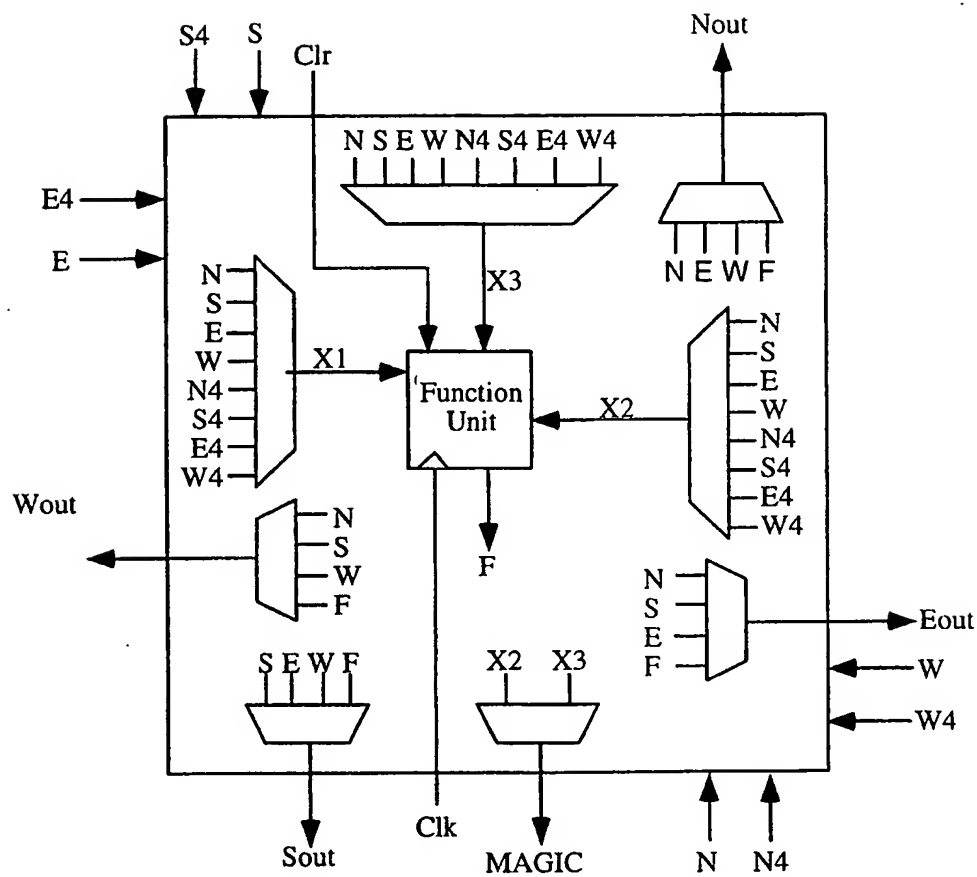


FIG. 3A  
(Prior Art)

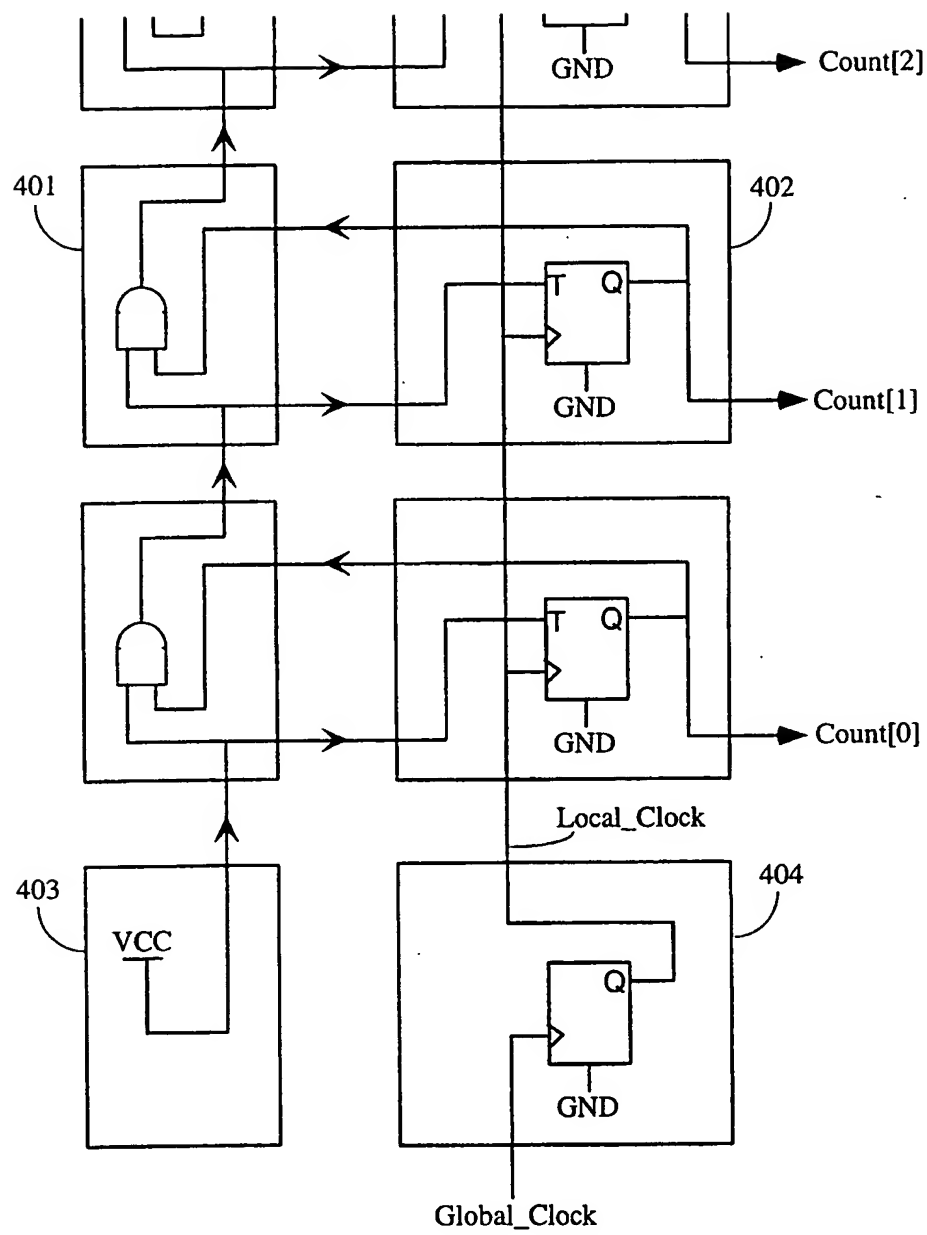


FIG. 4

```
Pci6200 pci6200 = new Pci6200N(null); // Hardware interface
pci6200.connect();
Register counterReg =      new Register(COLUMN, counterMap, pci6200);
Register clockReg =        new Register(COLUMN, clockMap, pci6200);
ClockMux localClock =      new ClockMux(ClockMux.CLOCK_IN);
ClockMux globalClock =     new ClockMux(ClockMux.GLOBAL_CLOCK);
ClearMux clear =           new ClearMux(ClearMux.ZERO);
Logic tff =                new Logic(Logic.T_FLIP_FLOP, Logic.EAST);
Logic clock =              new Logic(Logic.REGISTER);
Logic one =                new Logic(Logic.ONE);
Logic carry =              new Logic(Logic.AND, Logic.NORTH, logic.WEST);
carry.setEastOutput(Logic.NORTH); // Set carry output
```

FIG. 5

6/7

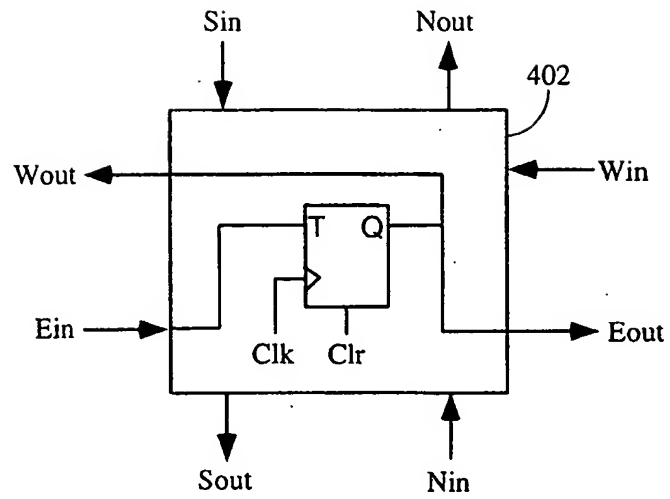


FIG. 6A

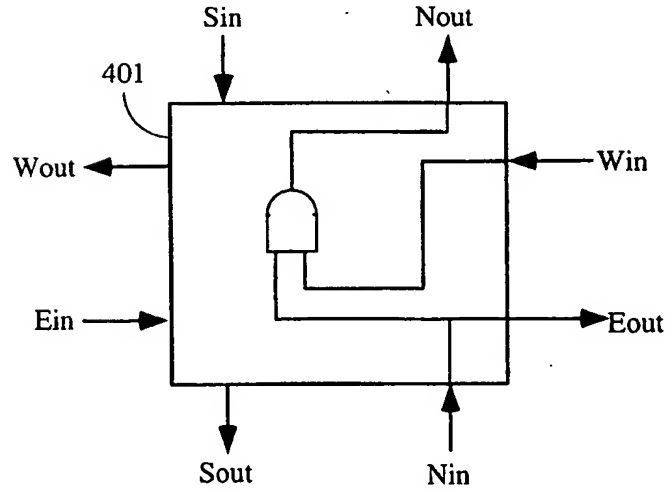


FIG. 6B



```
/* Configure cells */
for (i=ROW_START; i<ROW_END; i++) { // The counter
    carry.write((COLUMN-1), i, pci6200);
    tff.write(COLUMN, i, pci6200);
    localClock.write(COLUMN, i, pci6200);
    clear.write(COLUMN, i, pci6200);
} /* end for */
one.write((COLUMN-1), (ROW_START-1), pci6200); // Carry in
clock.write(COLUMN, (ROW_START-1), pci6200); // Clock
localClock.set(ClockMux.NORTH_OUT);
localClock.write(COLUMN, ROW_START, pci6200);
globalClock.write(COLUMN, (ROW_START-1), pci6200);
```

FIG. 7

```
for (i=0; i<5; i++) {
    clockReg.set(0); // Toggle clock
    clockReg.set(1);
    System.out.println("Count: " + counterReg.get());
} /*end for() */
```

FIG. 8A

```
C: \java\JERC> java Counter
Count: 0
Count: 1
Count: 2
Count: 3
Count: 4
C: \java\JERC>
```

FIG. 8B

# INTERNATIONAL SEARCH REPORT

International Application No

PCT/US 98/16436

**A. CLASSIFICATION OF SUBJECT MATTER**  
IPC 6 G06F17/50

According to International Patent Classification (IPC) or to both national classification and IPC

**B. FIELDS SEARCHED**

Minimum documentation searched (classification system followed by classification symbols)  
IPC 6 G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

**C. DOCUMENTS CONSIDERED TO BE RELEVANT**

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	WO 94 10627 A (GIGA OPERATIONS CORP ;TAYLOR BRAD (US); DOWLING ROBERT (US)) 11 May 1994 see page 8, line 5 - line 11 see page 35, line 6 - page 43, line 2 see page 45, line 1 - line 8; figures 17-30 ---	1-14
A	US 5 499 192 A (KNAPP STEVEN K ET AL) 12 March 1996 see column 3, line 53 - column 4, line 37 ---	1-14
A	EP 0 645 723 A (AT & T CORP) 29 March 1995 see page 3, line 29 - line 39 see page 4, line 9 - line 13 ---	1-14
	-/--	

☒ Further documents are listed in the continuation of box C.

☒ Patent family members are listed in annex.

\* Special categories of cited documents :

- "A" document defining the general state of the art which is not considered to be of particular relevance
- "E" earlier document but published on or after the international filing date
- "L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)
- "O" document referring to an oral disclosure, use, exhibition or other means
- "P" document published prior to the international filing date but later than the priority date claimed

- "T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
- "X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
- "Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.
- "&" document member of the same patent family

Date of the actual completion of the international search

8 December 1998

Date of mailing of the international search report

28/12/1998

Name and mailing address of the ISA  
European Patent Office, P.B. 5818 Patentlaan 2  
NL - 2280 HV Rijswijk  
Tel. (+31-70) 340-2040, Tx. 31 651 epo nl,  
Fax: (+31-70) 340-3016

Authorized officer

Guingale, A

## INTERNATIONAL SEARCH REPORT

International Application No

PCT/US 98/16436

## C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
P,X	<p>LECHNER E ET AL: "The Java Environment for Reconfigurable Computing" FIELD-PROGRAMMABLE LOGIC AND APPLICATIONS. 7TH INTERNATIONAL WORKSHOP, FPL '97. PROCEEDINGS, FIELD-PROGRAMMABLE LOGIC AND APPLICATIONS. 7TH INTERNATIONAL WORKSHOP, FPL '97. PROCEEDINGS, LONDON, UK, 1-3 SEPT. 1997, pages 284-293, XP002086682 ISBN 3-540-63465-7, 1997, Berlin, Germany, Springer-Verlag, Germany see the whole document -----</p>	1-14

# INTERNATIONAL SEARCH REPORT

information on patent family members

International Application No

PCT/US 98/16436

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
WO 9410627 A	11-05-1994	US 5535342 A	09-07-1996
		AU 5458194 A	24-05-1994
		CA 2148813 A	11-05-1994
		EP 0746812 A	11-12-1996
		JP 8504285 T	07-05-1996
		AU 5593594 A	24-05-1994
		CA 2148814 A	11-05-1994
		EP 0667010 A	16-08-1995
		JP 8504514 T	14-05-1996
		WO 9410624 A	11-05-1994
		US 5497498 A	05-03-1996
		US 5603043 A	11-02-1997
US 5499192 A	12-03-1996	NONE	
EP 0645723 A	29-03-1995	CA 2126265 A	28-03-1995
		JP 7152806 A	16-06-1995
		US 5594657 A	14-01-1997